Programación Python para Machine Learning

Lección 12: Modelos no supervisados.

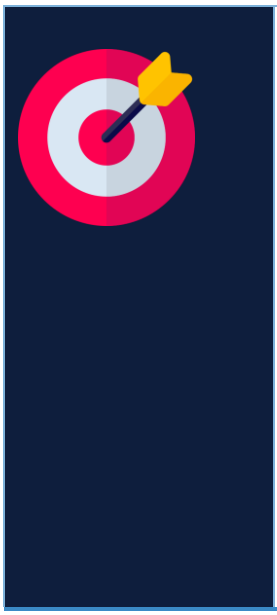
eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 12: Modelos no supervisados.

1. Introducción

Las unidades anteriores han sido destinadas a conocer técnicas de Machine Learning supervisadas, en las que se partía de patrones los cuales disponían de una variable objetivo con valores específicos que era utilizada para entrenar los modelos. Sin embargo, en algunos casos no se dispone de datos etiquetados, cambiando completamente la perspectiva del problema. Esta lección presenta diferentes modelos no supervisados y cómo pueden ser implementados en Python.



Objetivos

- Conocer los principios en los que se basan y la utilidad de los métodos no supervisados de Machine Learning
- Analizar las distintas variantes de métodos no supervisados.
- Dominar las técnicas de implementación de los métodos no supervisados en Python.
- Interpretar los resultados de aplicar técnicas no supervisadas a conjuntos de datos.

2. Principios teóricos de los métodos de Clustering

Las técnicas de Machine Learning no supervisado son modelos que se orientan fundamentalmente a analizar la estructura intrínseca y común de los datos. Su propósito puede ser, por una parte, la reducción de la dimensionalidad como, por ejemplo, el Análisis de Componentes Principales, o bien por otra, la agrupación o clustering de patrones.

En esencia, el objetivo de las **técnicas de clustering** es buscar diferentes grupos dentro de las instancias de un conjunto datos. Para ello, los algoritmos de este tipo analizan las estructuras en los datos de manera que los elementos del mismo grupo (clúster) sean lo más similares entre sí y que los diferencie del resto de grupos.

Aunque hay más métodos, en este caso se van a presentar dos de los conjuntos de métodos de Clustering más populares:

- Métodos de clustering particional.
- Métodos de clustering jerárquico.

2.1. Métodos de clustering particional

Los algoritmos de clustering particional tiene como objetivo es obtener una partición de las instancias en clústeres de tal forma que todas ellas acaben perteneciendo a alguno de los posibles clústeres y que, a su vez, los clústeres sean lo más distantes posibles. Es decir, son métodos que se basan en agrupar las instancias que tienen una alta similitud entre ellas, cuantificando tal similitud en términos de distancia. De este modo, la probabilidad de pertenecer al mismo clúster aumenta cuanto más cerca (más similares) estén las instancias.

Existen varios métodos de clustering particional. Sin embargo, el más popular es el algoritmo iterativo denominado ***K-means***, que sigue la siguiente estructura:

1. Considerar los aleatoriamente *k* elementos del conjunto de datos y considerarlos como centroides de clústeres, asignando el resto de los patrones al centroide más próximo.
2. Recalcular el nuevo centroide de todos los clústeres.
3. Reasignar cada instancia al clúster del centroide más cercano.
4. Repetir los pasos 2 y 3 hasta alcanzar un determinado criterio de parada.

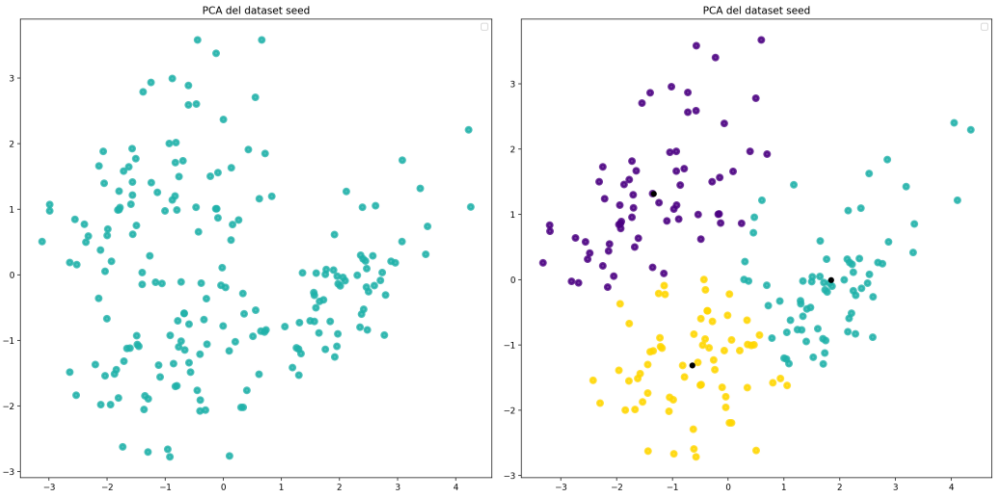


Figura 2.1: Gráfico de las dos primeras componentes de un PCA sobre el resultado de un k-means sobre el dataset Seeds con k=3.

La figura muestra una representación gráfica de los dos primeros componentes de un PCA sobre el resultado de aplicar un k-means con *k=3*, coloreando las instancias por el clúster en el que han sido incluidas.

En este punto surgen dos aspectos a tener en cuenta a la hora de poner en práctica este procedimiento. Por un lado, qué **distancia** considerar, y por otro, el **número de clústeres *(k)*** a tener en cuenta más adecuado.

De modo general la distancia utilizada es la distancia euclídea, aunque este algoritmo permite utilizar cualquier distancia entre patrones (Manhattan, Hamming, correlación de Pearson...). En cualquier caso, hay que considerar si para la distancia considerada las características deben medirse en la misma escala o no, por lo que puede ser necesario realizar un proceso de estandarización adecuado.

La elección del número adecuado de clústeres es uno de los puntos más sensibles de este algoritmo. Es posible determinar este número a priori si se tiene un conocimiento del problema. Otra opción sería hacer uso de un método analítico que soporte con datos la elección del valor. El método del codo es una de los métodos de este tipo más populares.

El **método del codo** traza gráficamente una curva los valores ascendentes de *k* frente al porcentaje de varianza obtenida al usar esa *k* en el algoritmo. El objetivo es encontrar la *k* adecuada para que en cada clúster no aumente significativamente la varianza.

Una de las características de estos métodos de clustering es lo extremadamente sencillos que son de implementar, y simultáneamente, tienen una alta eficiencia computacional. Por el contrario, presentan serias dificultades a la hora de tratar con datos que tienen determinadas distribuciones.

2.2. Métodos de clustering jerárquico

El clustering jerárquico es un grupo de técnicas algorítmicas de Machine Learning no supervisado utilizadas para agrupar instancias de conjuntos de datos. El objetivo del clustering jerárquico, al igual que el clustering particional, es el de agrupar los patrones con características similares. De hecho, en ciertos casos, al aplicar una técnica de clustering jerárquica y una de clustering particional puede surgir un resultado parecido.

Se pueden considerar dos tipos de técnicas de clustering jerárquico:

- Algoritmo aglomerativo: los grupos se conforman utilizando un enfoque ascendente partiendo de la consideración de las instancias individualmente.
- Algoritmo divisivo: se basan en un enfoque descendente en el que todos los puntos de datos se tratan como un gran clúster y el proceso de agrupación implica la división del gran clúster en varios clústeres pequeños.

Aunque la similitud entre patrones puede cuantificarse mediante cualquiera de las distancias expuestas anteriormente, en este tipo de técnicas, la distancia entre clústeres puede medirse según varias opciones:

- Entre los puntos más cercanos de dos clústeres.
- Entre los puntos más lejanos de dos conglomerados.
- Entre los centros de dos conglomerados.
- Entre todas las combinaciones posibles de puntos entre los dos clústeres y tomar la media.

El algoritmo aglomerativo seguiría la siguiente secuencia:

1. Calcular la matriz de proximidad entre instancias.
2. Considerar cada instancia como un clúster.
3. Combinar los dos clústeres más cercanos.
4. Actualizar la matriz de proximidad.
5. Repetir los puntos 3 y 4 hasta que quede un solo clúster.

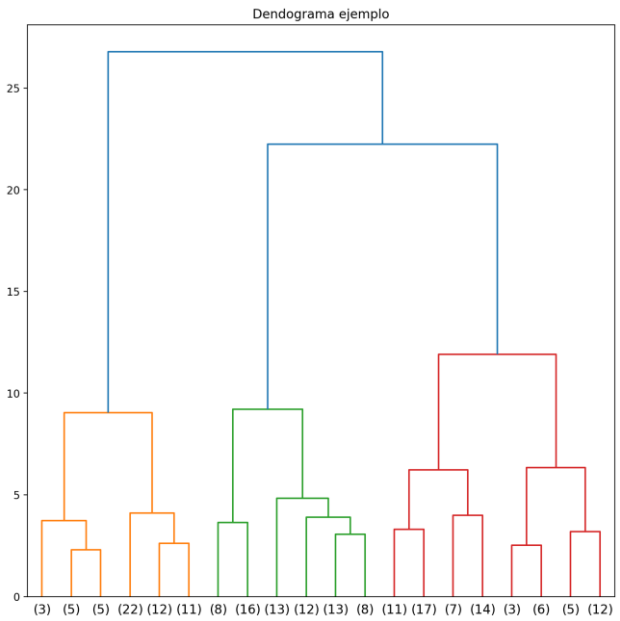


Figura 2.2: Dendograma ejemplo con 20 clústeres finales.

Una vez se aplica este tipo de técnica jerárquica, es muy interesante observar gráficamente el resultado. Para ello existe un tipo de gráfica denominada dendograma.

Un **dendrograma** es un diagrama en forma de árbol que organiza los datos en subcategorías que se van dividiendo en otros hasta llegar al nivel de detalle deseado. Este tipo de representación permite apreciar claramente las relaciones de agrupación entre los datos e incluso entre grupos de ellos, aunque no las relaciones de similitud o cercanía entre categorías.

Como puede verse en el ejemplo de la figura anterior**¡Error! No se encuentra el origen de la referencia.**, observando las sucesivas subdivisiones es posible hacerse una idea sobre los criterios de agrupación de los mismos, la distancia entre los datos según las relaciones establecidas, etc.

3. Implementación en Python de un método de clustering particional: K-means

K-means es una técnica de Machine Learning no supervisado que por la sencillez y el rendimiento que presenta la hacen ser un modelo apropiado para resolver problemas de clustering.

En Python es posible implementar K-means utilizando la clase `KMeans` dentro del módulo `cluster` de scikit-learn. Por defecto, `KMeans` tiene como número de clúster a buscar establecido en 8, aunque es posible cambiar esta configuración utilizando el parámetro `n_clusters`. Además, una vez que termina el proceso, será posible utilizar varios atributos de la clase como la posición de los centroides `cluster_centers_`, el cluster donde ha sido categorizado cada instancia labels_ y la suma total de las distancias de las instancias a su centroide `inertia_`. Se hará uso del conjunto de datos *seeds* del repositorio *UCI Machine Learning* para ilustrar el proceso.

```
from sklearn.cluster import KMeans
from pandas import read_csv
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

filename = '../datasets/seeds.data'

col_names=['Area', 'Per', 'Comp', 'LenK', 'Width', 'Asy', 'LenKG']
data = read_csv(filename, names=col_names,sep="\t")

data.dropna(inplace=True)
X= data
X= data
scaler =StandardScaler()
X_train = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

print(kmeans.cluster_centers_)
print(kmeans.labels_)

#método del codo
lk=[]
for k in range(1,10):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    lk.append(kmeans.inertia_)
plt.plot( range(1,10),lk)
```

4. Implementación en Python de un clustering jerárquico aglomerativo

El clustering aglomerativo es una técnica de Machine Learning no supervisado apropiada para resolver problemas de agrupamiento bajo los principios del clustering jerárquico.

En Python es posible hacer uso implementación que tiene scikit-learn de esta técnica utilizando la clase `AgglomerativeClustering` dentro del módulo `cluster`. Por defecto, `AgglomerativeClustering` tiene como número de clúster a buscar establecido en 2, aunque es posible cambiar esta configuración utilizando el parámetro `n_clusters`. Además, es posible determinar el criterio que considera el algoritmo a optimizar utilizando el parámetro `linkage`, que por defecto considera el valor `'ward'`, opción que trata de minimizar la varianza entre clústeres.

Una vez que termina el proceso, será posible utilizar varios atributos de la clase como la posición de los centroides `cluster_centers_`, el clúster donde ha sido categorizado cada instancia `labels_`.

Por otra parte, Scipy dentro de su módulo paquete `cluster`, cuenta con el módulo `hierarchy`, que dispone de la función `dendogram` para construir un dendograma que ilustre el resultado del proceso.

Se hará uso del conjunto de datos *seeds* del repositorio *UCI Machine Learning* para ilustrar el proceso.

```
import matplotlib.pyplot as plt
from pandas import read_csv
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
```

```
filename = '../datasets/seeds.data'
col_names=['Area', 'Per', 'Comp', 'LenK', 'Width', 'Asy', 'LenKG']
data = read_csv(filename, names=col_names,sep="\t")

data.dropna(inplace=True)
X=data
scaler =StandardScaler()
X_train = scaler.fit_transform(X)

cluster = AgglomerativeClustering(n_clusters=5, linkage='ward')
cluster.fit_predict(X)
print(cluster.labels_)

import scipy.cluster.hierarchy as dnd
plt.figure()
plt.title("Customer Dendograms")
dend = dnd.dendrogram(dnd.linkage(X, method='ward'))
```

5. CONSEJOS PRÁCTICOS SOBRE LOS MÉTODOS NO SUPERVISADOS

A continuación, se aportan una serie de recomendaciones a la hora de poner en práctica cualquier tipo de técnica no supervisada de Machine Learning:

- **Normalización o estandarización de datos:** es buena idea que los valores de cada característica estén en escalas similares. Un proceso previo de normalización o de estandarización de datos ayudará al algoritmo de clustering puesto que los grupos se forman a partir de distancias. Hay que evitar que haya características con escalas muy diferentes debido a que las características de mayor escala dominarán las distancias.
- **Número de clústeres:** hay las técnicas de clustering, por ejemplo, K-means, necesitan que se le especifique el número de clústeres. No es obvio, a priori, saber qué número de grupos es el óptimo. Como se ha mencionado, es posible determinar este número a priori si se tiene un conocimiento previo del problema. Otra opción sería hacer uso de un método analítico que soporte con datos la elección del valor. El método del codo es una de los métodos de este tipo más populares.

6. PUNTOS CLAVE

En esta lección se ha aprendido:

- Conocer de dónde surgen, los principios en los que se basan y la utilidad de los métodos no supervisados de Machine Learning.
- Analizar las distintas variantes de métodos no supervisados: los modelos particionales y los jerárquicos
- Implementar en scikit-learn un modelo de Kmeans para problemas de clustering.
- Implementar en scikit-learn un modelo de aglomerativo para problemas de clustering.
- Interpretar los resultados obtenidos tras aplicar técnicas no supervisadas a conjuntos de datos.

https://towardsdatascience.com/bagging-on-low-variance-models-38d3c70259db

https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe

https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/